Introduction to Scientific Python

CCNSS 2016

Part II

Topics Part II

Functions

Topics Part II

Functions

Modules

Topics Part II

Functions

Modules

Packages

Topics Part II

Functions

Modules

Packages

Advanced Topics

Functions

Functions

```
def mysum(a, b):
  '''Return a + b'''
  return a + b
```

Functions

```
def mysum(a, b):
  '''Return a + b'''
  return a + b

print mysum(1, 2)
```

```
3
```

Functions

```
def mysum(a, b):
  '''Return a + b'''
  return a + b


print mysum(1, 2)
```

```
3
```

```
help(mysum)
```

```
Help on function mysum in module __main__:

mysum(a, b=2)
 Return a + b
```

Functions

Call by argument names

```
print mysum(a=1, b=2)
```

```
3
```

Functions

Call by argument names

```
print mysum(a=1, b=2)
```

```
3
```

Mandatory arguments vs default values

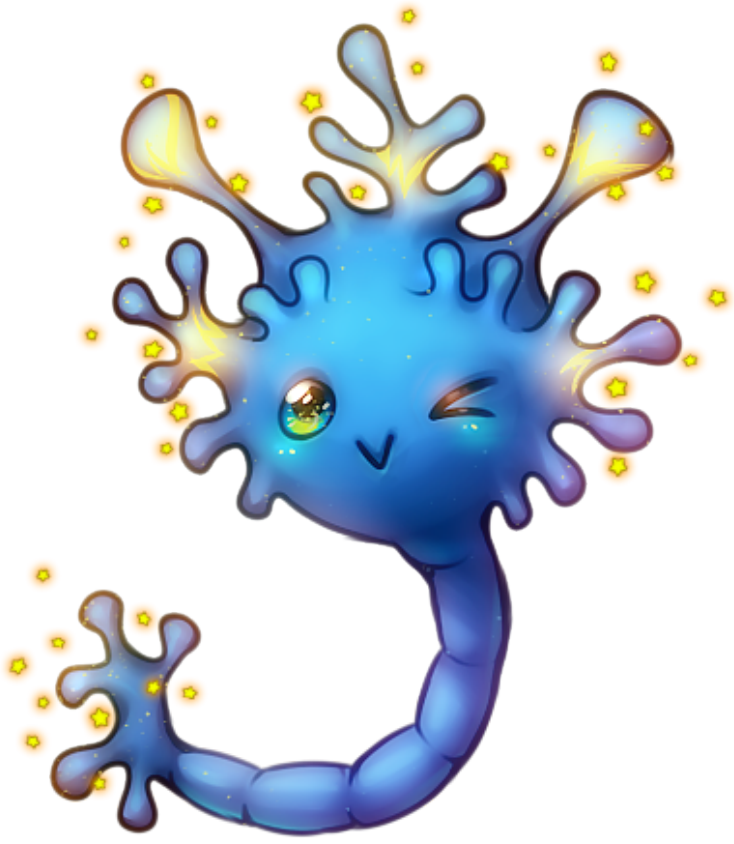
```
def mysum(a, b=2):
  '''Return a + 2 or a + b'''
  return a + b

print mysum(1)
```

```
3
```

Functions Notebook

LIF Neuron Exercise



Coding Time!

(Exercise 11)

Use functions

Modules

Modules

```
def mysum(a, b=2):
  '''Return a + 2 or a + b'''
  return a + b
```

Save as file mymath.py

Modules

```
def mysum(a, b=2):
  '''Return a + 2 or a + b'''
  return a + b
```

Save as file mymath.py

```
import mymath

print mymath.mysum(1, 2)
```

```
3
```

Import Types

```
import mymath as mm

print mm.mysum(1, 2)
```

```
3
```

Import Types

```
import mymath as mm

print mm.mysum(1, 2)
```

```
3
```

```
from mymath import mysum

print mysum(1, 2)
```

```
3
```

Import Types

```
import mymath as mm

print mm.mysum(1, 2)
```

```
3
```

```
from mymath import mysum

print mysum(1, 2)
```

```
3
```

```
from mymath import *

print mysum(1, 2)
```

```
3
```

Modules

Word of advice: **be explicit!**

Modules

Word of advice: **be explicit!**

`np.array, plt.plot`

...you'll get used to it.

Modules Notebook

Packages

Numpy

Numpy

Fundamental package for scientific computing

Numpy

Fundamental package for scientific computing

Linear algebra, Fourier transform and random numbers

Numpy

Fundamental package for scientific computing

Linear algebra, Fourier transform and random numbers

N-dimensional `array` object

Numpy

Fundamental package for scientific computing

Linear algebra, Fourier transform and random numbers

N-dimensional `array` object

Broadcasting functions

Numpy

Fundamental package for scientific computing

Linear algebra, Fourier transform and random numbers

N-dimensional `array` object

Broadcasting functions

Integrate C/C++ and Fortran code

Scipy

Partner of Numpy package

Scipy

Partner of Numpy package

Fundamental library for scientific computing 

Scipy

Partner of Numpy package

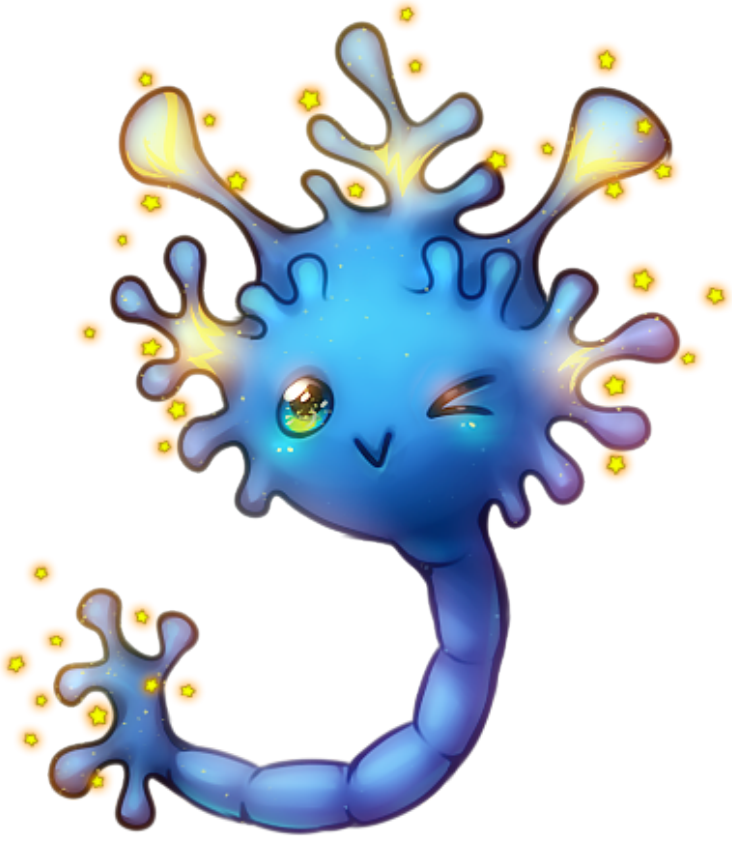
Fundamental library for scientific computing 

Special functions
Integration
Optimization
Interpolation
Signal Processing
Statistics
Multidimensional image processing
...

Packages

Notebook

LIF Neuron Exercise



Coding Time!

(Exercise 12)

Use NumPy

Advanced Topics

Standard Variable Types - Tuples

Tuples are read-only lists

```
mytuple = (100, 1000.0, "John", 0.5 + 0.5j)
print mytuple[0:1]
```

Standard Variable Types - Tuples

Tuples are read-only lists

```
mytuple = (100, 1000.0, "John", 0.5 + 0.5j)
print mytuple[0:1]
```

```
(100,)
```

Standard Variable Types - Tuples

Tuples are read-only lists

```
mytuple = (100, 1000.0, "John", 0.5 + 0.5j)
print mytuple[0:1]
```

```
(100,)
```

```
print mytuple[0:1][0]
```

Standard Variable Types - Tuples

Tuples are read-only lists

```
mytuple = (100, 1000.0, "John", 0.5 + 0.5j)
print mytuple[0:1]
```

```
(100,)
```

```
print mytuple[0:1][0]
```

```
100
```

Standard Variable Types - Sets

Return unique elements of lists and tuples

```
myset = set([1,1,2,3,4])
print myset
```

Standard Variable Types - Sets

Return unique elements of lists and tuples

```
myset = set([1,1,2,3,4])
print myset
```

```
set([1, 2, 3, 4])
```

List Comprehensions

One-liner `for` loops

```
squares = []
for x in range(5):
    squares += [x**2]
print squares
```

List Comprehensions

One-liner `for` loops

```
squares = []
for x in range(5):
    squares += [x**2]
print squares
```

```
[0, 1, 4, 9, 16]
```

List Comprehensions

One-liner `for` loops

```
squares = []
for x in range(5):
    squares += [x**2]
print squares
```

```
[0, 1, 4, 9, 16]
```

```
squares = [x**2 for x in range(10)]
print squares
```

List Comprehensions

One-liner `for` loops

```
squares = []
for x in range(5):
    squares += [x**2]
print squares
```

```
[0, 1, 4, 9, 16]
```

```
squares = [x**2 for x in range(10)]
print squares
```

```
[0, 1, 4, 9, 16]
```

Enumerate Construct

Returning indexes and elements

```
mylist = ['pyramidal', 'inhibitory', 'glial']
for idx, item in enumerate(mylist):
    print idx, item
```

Enumerate Construct

Returning indexes and elements

```
mylist = ['pyramidal', 'inhibitory', 'glial']
for idx, item in enumerate(mylist):
    print idx, item
```

```
0 pyramidal
1 inhibitory
2 glial
```

That's all folks!